GUIDELINES FOR SECURE SOFTWARE; DEVELOPING SECURITY PROTOCOLS

General Point

- All the usual system design guidelines apply
- Simple
- Modular
- No repetition
- Narrow interfaces

Is Software Development Like This?

Trusted Path

- Have a trusted path for entering security information
  - Login screens
  - Entering PIN
  - Sending stuff by mail
- Ensure that you are talking directly to the user
  - Secure login sequence
  - Smartcard readers with PIN pad
  - https://www.mybank.com

Concurrency Issues

- Race conditions: two parallel processes try to act on the same resource
- updateBalance(account, amount)
  - oldBalance = account.getBalance()
  - newBalance = oldBalance + amount
  - account.setBalance(newBalance)
- If several threads try to update the same account, the result is unpredictable

TOCTOU error

- Time of check – time of use
- Check access – use resource
  - Check for file – write/read file
- Error in the middle of multi-step operation
- Be very careful when writing privileged programs
- Use atomic operations offered by the OS
- Re-check after using the resource

Sending Feedback

- Minimize output of technical info
- No version numbers in login prompt
- No stack traces etc. on errors
- Technical information can go to log files

Set Resource Limits

- Try to set limits on used resources
  - Memory
  - CPU
- Bad idea: spawn 1000 processes, each doing intensive calculations and IO
- OS can help in setting limits

Separate Data and Progams

- Do not use programs as config files
  - No include_once('settings.inc')
- Do not embed programs in data files
  - Word macro viruses
- In web apps, keep data and programs in different directories

Minimize Privileges

- Minimize privileges of the process
  - Do not run processes as system user
  - Create special user for the task
- Minimize time privilege can be used
  - Usual practice is to do privileged action and drop privileges

Minimize Privileges (2)

- Minimize the modules that have privilege
  - Concentrate high-privilege stuff in the TCB
- Consider sandboxes
  - chroot
  - Virtual machines

API Security

- Subject: API operating between low and high
  - OS API
  - Hardware device API
  - Hardware security module interface/protocol
- Keep the API minimal
  - More features mean more code and more bugs, more unexpected interactions
  - What you publish, you must maintain later

API Security (2)

- Export opaque handles
  - Keep the objects inside API
  - In high-security cases, export integer handles instead of pointers
- Have safe defaults
  - Do not export unencrypted secrets
- Remember: trustno1
  - Calling code can be malicious
  - Always check everything

API Security (3)

- Be explicit about states
  - Create state machine model for API objects
  - Check that operations are called in the correct states
- Enforce correctness via type system
  - Have things like „key”, „encrypted key”
  - Separate keys for different purposes (encryption, signing, key encryption)

API Security (4)

- Make it easy to write correct programs
  - Prevent TOCTOU errors
- Beware of several APIs by the same component
- API interaction
  - Browser limits functinality available to JavaScript
  - However, plugins can provide APIs with more features

API Security (5)

- Concurrency issues
  - API can be called from several threads
  - Beware of races inside the API implementation
  - Example: operating system wrappers

API Design Reading

- Peter Gutmann, „The Design of a Cryptographic Security Architecture”
  - http://www.cs.auckland.ac.nz/~pgut001/pubs/usenix99.pdf

Designing Protocols

Explicitness Principle

- Messages should contain explicit information
- Who is sender?
- Who is receiver?
- Field lengths, separators, types
- Algorithms used
- What are content types
- Put important stuff under signature
  - Hash of list of important values

ASN.1

- Abstract Syntax Notation One
- Data encoding, used by several crypto protocols
  - Digital certificates
  - PKI protocols
  - S/MIME (secure e-mail)
- Binary format
- TLV (tag-length-value)

ASN.1 Data Types

- BOOLEAN
- INTEGER (unlimited length)
- BIT STRING
- OCTET STRING
- NULL (absence of an element)
- OBJECT IDENTIFIER (unique identifier, hierarchical)
- STRING

ASN.1 Data Types (2)

- SEQUENCE (sequence of fields, like *struct*)
- SEQUENCE OF (sequence of same type, like list)
- SET (unordered sequence)
- SET OF (same)
- CHOICE (one of the fields, like *union*)
- ANY (can contain any ASN.1 type)

ASN.1 Example

```
TimestampRequest ::= SEQUENCE {
    version          INTEGER { v1(1) },
    messageImprint MessageImprint,
}
MessageImprint ::= SEQUENCE {
    hashAlgorithm AlgorithmIdentifier,
    hashedMessage OCTET STRING
}
AlgorithmIdentifier  ::=  SEQUENCE  {
    algorithm  OBJECT IDENTIFIER,
    parameters ANY DEFINED BY algorithm OPTIONAL
}
```

ASN.1 Example (2)

```
 0 30  38: SEQUENCE {
 2 02   1:   INTEGER 1
 5 30  33:   SEQUENCE {
 7 30   9:     SEQUENCE {
 9 06   5:       OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
16 05   0:       NULL
          :     }
18 04  20:     OCTET STRING
          :       80 25 6F 39 A9 D3 08 65 0A C9 0D 9B E9
A7 2A 95
          :       62 45 45 74
          :     }
          : }
```

More About ASN.1

- Several ways of encoding data
  - Basic encoding rules
  - Distinguished encoding rules
  - XML encoding rules
- DER encodes the same data always the same way
- BER is easier to process in some situations
- Tools for compiling message descriptions

Creating Your Own Protocol

- Two independent implementors must be able to create compatible implementation
- Your task is to help them
- Make it simple
- Make it clear

Creating Your Own Protocol (2)

- „The question is not whether it is possible to imagine a situation where this feature could be useful, but rather are there real-life problems which require using this non-standard feature”
- Only include stuff that you really need
- … now
  - If you do not need it now, you are not going to need it later
  - You will not guess how you really want to implement it

Options

- Optional features are bad
- Choices are bad
  - The sender does not know what the receiver can process
  - What can he rely on?
  - What service will he get?
  - Same goes for receiver
- Do the optional bits change interpretation of the message?

Options (2)

- X.509 way: extensions can be marked critical
  - If you receive critical extension that you cannot process, you must not accept the message
- Feature negotiation
  - Possible, but usually complicated and requires sending several messages
- Always specify mandatory algorithms
- Fix character set
  - Recommended: UTF8

Options (3)

- Leave room for extension
- Identify signing/hashing/encryption algorithm by ID
- Use extensions/attributes
  - Extension = ID + data
  - Signed extensions
  - Unsigned extensions

Relating to Standards

- Standards are good
- For innovative systems, it is often impossible to conform to standards
- Standards compatibility is binary
  - You either are compatible or not
  - If other implementations cannot process messages correctly, you are not compatible

Relating to Standards (2)

- Do not mangle standards
  - Do not change fields
  - Do not change meanings of fields
- If you change field then you are not compatible
  - You might as well design completely new protocol

What to Specify

- Message formats
- How to compose message
- How to verify message
- Example messages

Specification Language

- Use language specified in RFC 2119/BCP 14
- MUST, REQUIRED, SHALL – absolute requirement
- MUST NOT, SHALL NOT – absolute prohibition
- SHOULD, SHOULD NOT – implementor may do differently, if he has good reasons
- MAY – implementor can do whatever he wants

Tidbits

- Use HTTP for transport
  - Lot of implementations
  - Provides flow control
  - Passes through firewalls
  - You do not have to create specification
- If you have trouble documenting it, it usually is not right

Tidbits (2)

- Machine should be able to make binary decision
  - Trust or no trust
  - Do not leave stuff for policy
  - Do not give imprecise assurance
- Creating XML-based security protocol can be difficult
  - No deterministic representation
  - You can use canonicalization